(12) UK Patent Application (19) GB (11) 2 378 015 (13) A

(43) Date of A Publication 29.01.2003

(21) Application No 0129835.5

(22) Date of Filing 13.12.2001

(30) Priority Data
(31) 09912389 (32) 26.07.2001 (33) US

(51) INT CL[7]
G06F 1/00

(52) UK CL (Edition V)
G4A AAP A23X

(56) Documents Cited
GB 2365158 A US 5991714 A

(58) Field of Search
UK CL (Edition T) G4A AAP AFMX
INT CL[7] G05F 1/00
Other: Online : WPI,EPODOC,PAJ,INSPEC, ELSEVIER,TDB

(71) Applicant(s)
Networks Associates Technology Inc
(Incorporated in USA - Delaware)
3965 Freedom Circle, Santa Clara,
CA 95054, United States of America

(72) Inventor(s)
Neil Andrew Cowie
Igor Garrievich Muttik

(74) Agent and/or Address for Service
D Young & Co
21 New Fetter Lane, LONDON, EC4A 1DA,
United Kingdom

(54) Abstract Title
Detecting computer programs within packed computer files

(57) A technique for detecting Trojans and worms within packed computer files uses fingerprint data derived from the unpacked resource data associated with the packed computer files. The number of entries, the position within the resource data and size of the resource that is the largest resource specified, a timestamp value of compilation and a checksum value derived from the whole of the resource data may be included within a fingerprint value as characteristic of a particular set of resource data. A library of such fingerprint values may be generated for known Trojans and worms, or other programs it is wished to detect, and then a suspect file compared against this library of fingerprints.

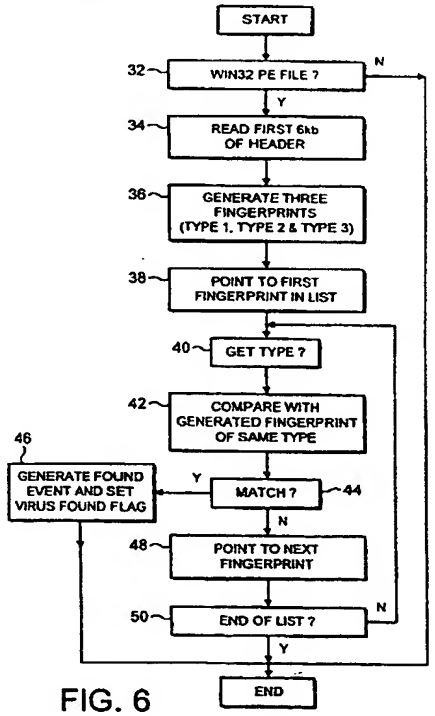


FIG. 6

GB 2 378 015 A

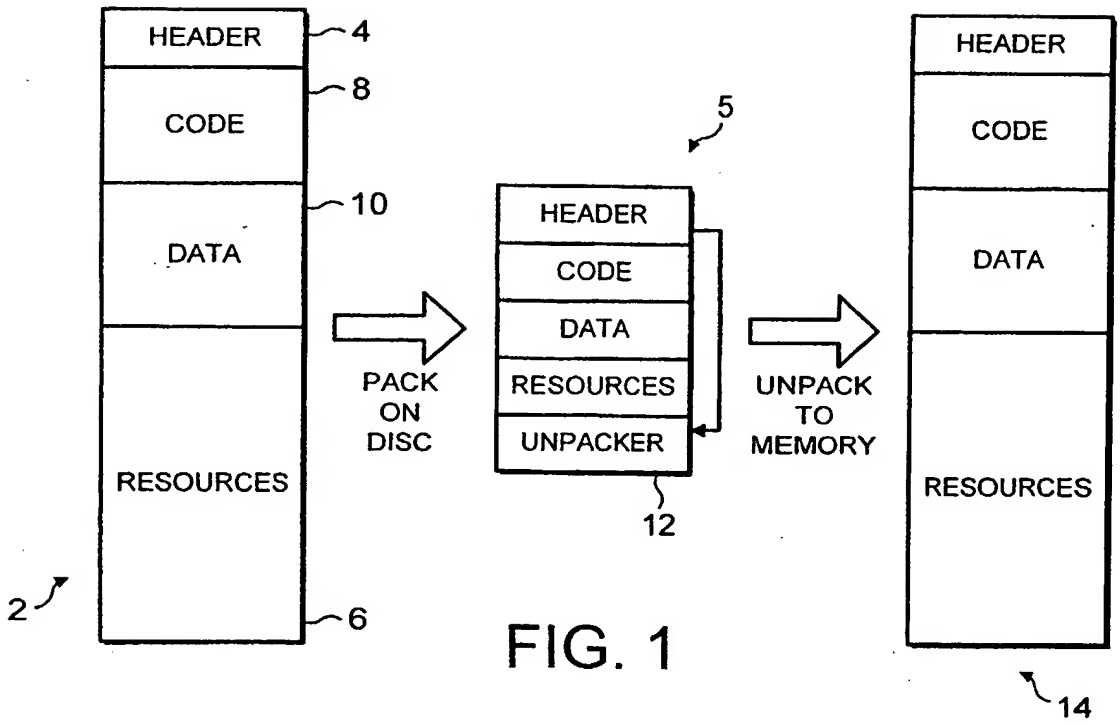

HEADER
CODE
DATA
RESOURCES
4
8
10
6
2
PACK
ON
DISC
5
HEADER
CODE
DATA
RESOURCES
UNPACKER
12
UNPACK
TO
MEMORY
HEADER
CODE
DATA
RESOURCES
14

FIG. 1

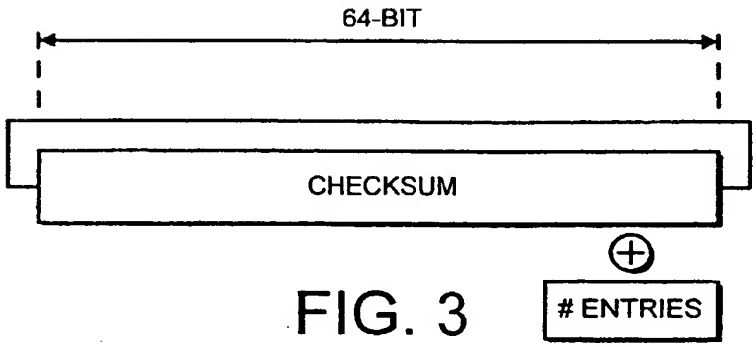

64-BIT
CHECKSUM
ENTRIES

FIG. 3

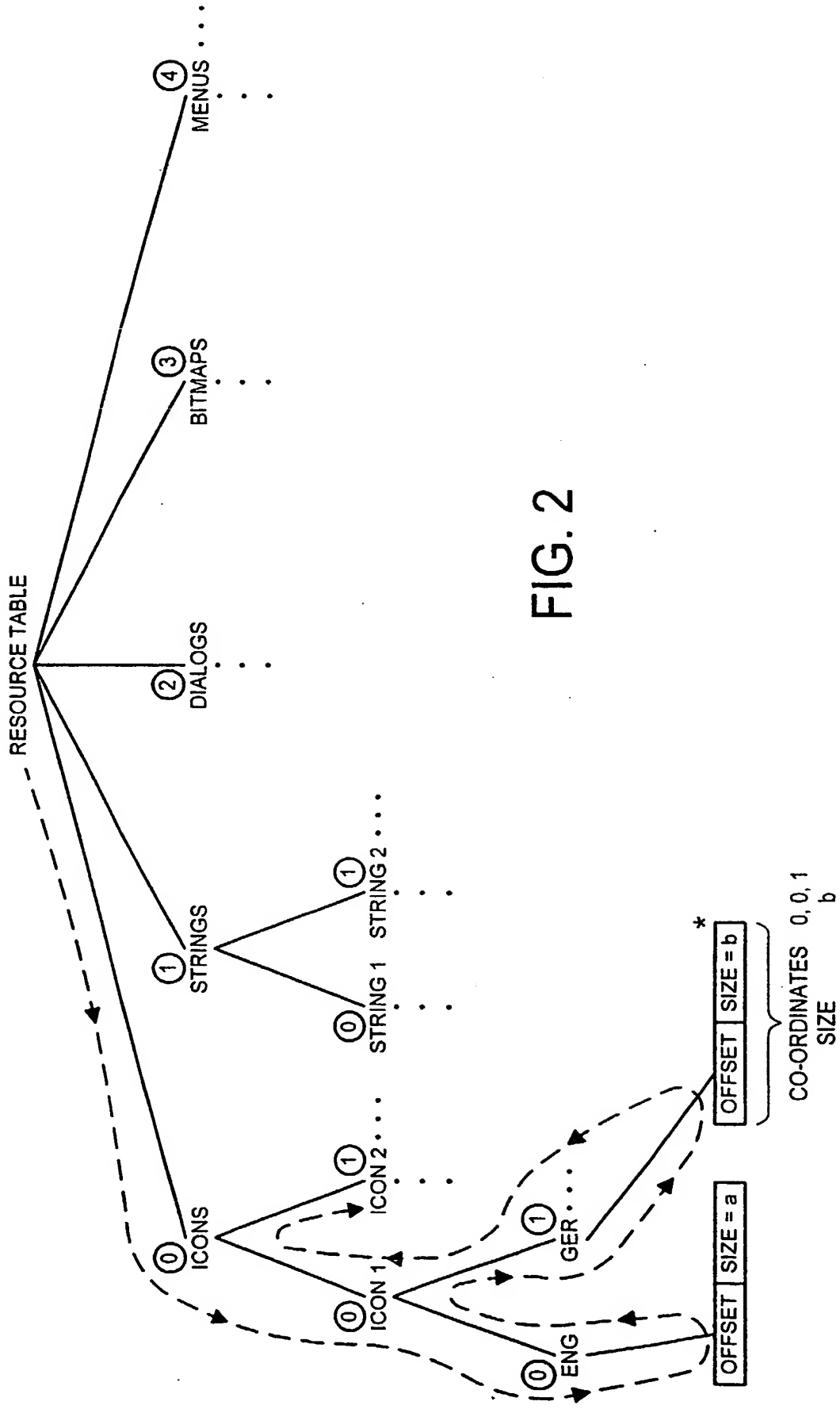
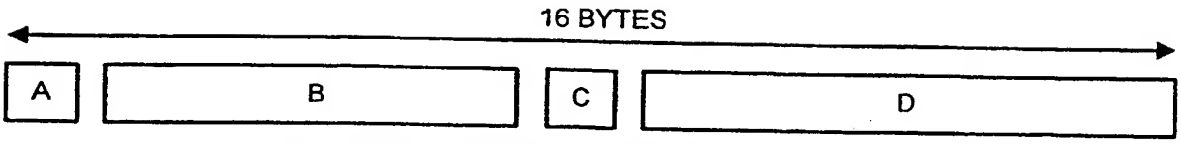

RESOURCE TABLE
0 ICONS
1 STRINGS
2 DIALOGS
3 BITMAPS
4 MENUS
0 ICON 1
1 ICON 2
0 STRING 1
1 STRING 2
0 ENG
1 GER
OFFSET
SIZE = a
OFFSET
SIZE = b
*
CO-ORDINATES 0, 0, 1
SIZE b

FIG. 2

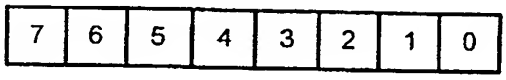
FINGERPRINT



A = 1 BYTE = CONTROL BYTE (SEE FIG. 4b)
B = 6 BYTES = ELIMINATION DATA (SEE FIG. 4c)
C = 1 BYTE = # ENTRIES PROCESSED
D = 8 BYTES = CHECKSUM

FIG. 4a

CONTROL BYTE



BITS 0 - 3 = RESERVED = 0
BITS 4 - 6 = 000 = RESERVED
001 = ELIMINATION DATA IS CO-ORD / SIZE
010 = ELIMINATION IS TIME STAMP
011 = ELIMINATION IS FILE LENGTH
100, 101, 110, 111 } RESERVED
BIT 7 = RESERVED = 0

FIG. 4b

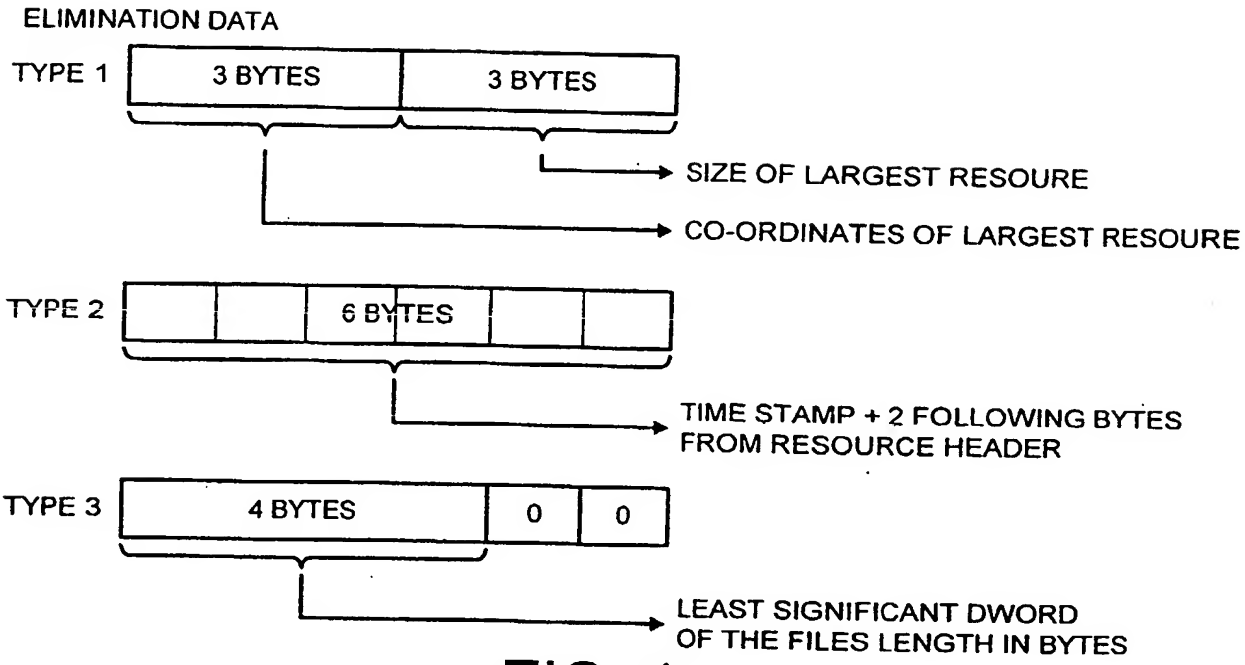

ELIMINATION DATA
TYPE 1
3 BYTES
3 BYTES
SIZE OF LARGEST RESOURE
CO-ORDINATES OF LARGEST RESOURE
TYPE 2
6 BYTES
TIME STAMP + 2 FOLLOWING BYTES
FROM RESOURCE HEADER
TYPE 3
4 BYTES
0
0
LEAST SIGNIFICANT DWORD
OF THE FILES LENGTH IN BYTES

FIG. 4c

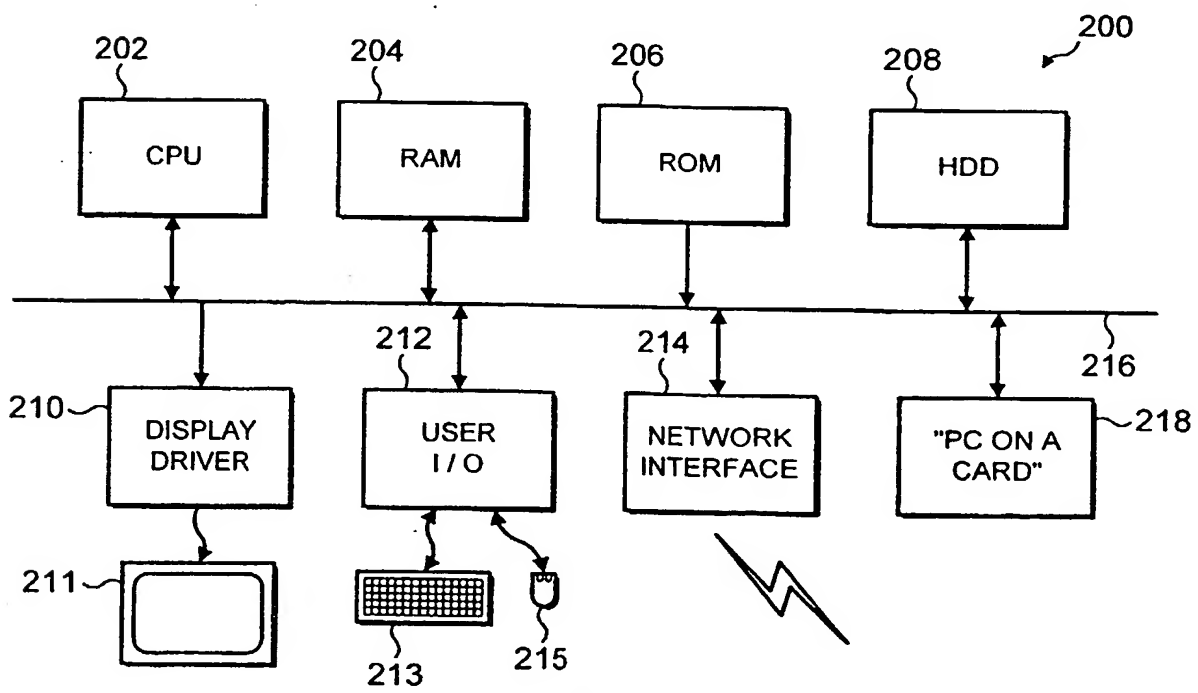

200
202
CPU
204
RAM
206
ROM
208
HDD
210
DISPLAY DRIVER
211
212
USER I/O
213
214
NETWORK INTERFACE
215
216
"PC ON A CARD"
218

FIG. 7

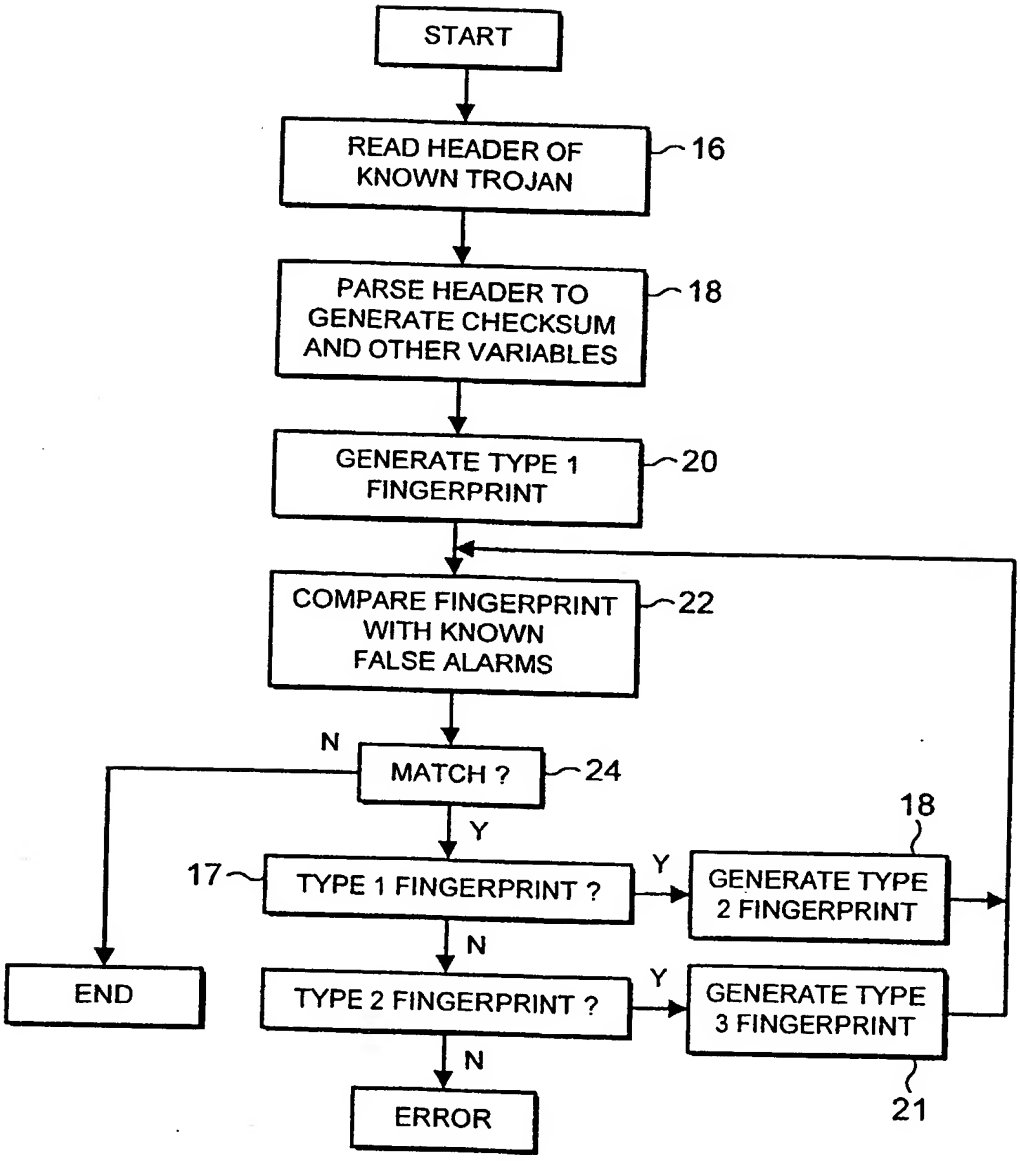

START
READ HEADER OF KNOWN TROJAN
16
PARSE HEADER TO GENERATE CHECKSUM AND OTHER VARIABLES
18
GENERATE TYPE 1 FINGERPRINT
20
COMPARE FINGERPRINT WITH KNOWN FALSE ALARMS
22
N
MATCH ?
24
Y
17
TYPE 1 FINGERPRINT ?
Y
18
GENERATE TYPE 2 FINGERPRINT
N
END
TYPE 2 FINGERPRINT ?
Y
GENERATE TYPE 3 FINGERPRINT
N
21
ERROR

FIG. 5

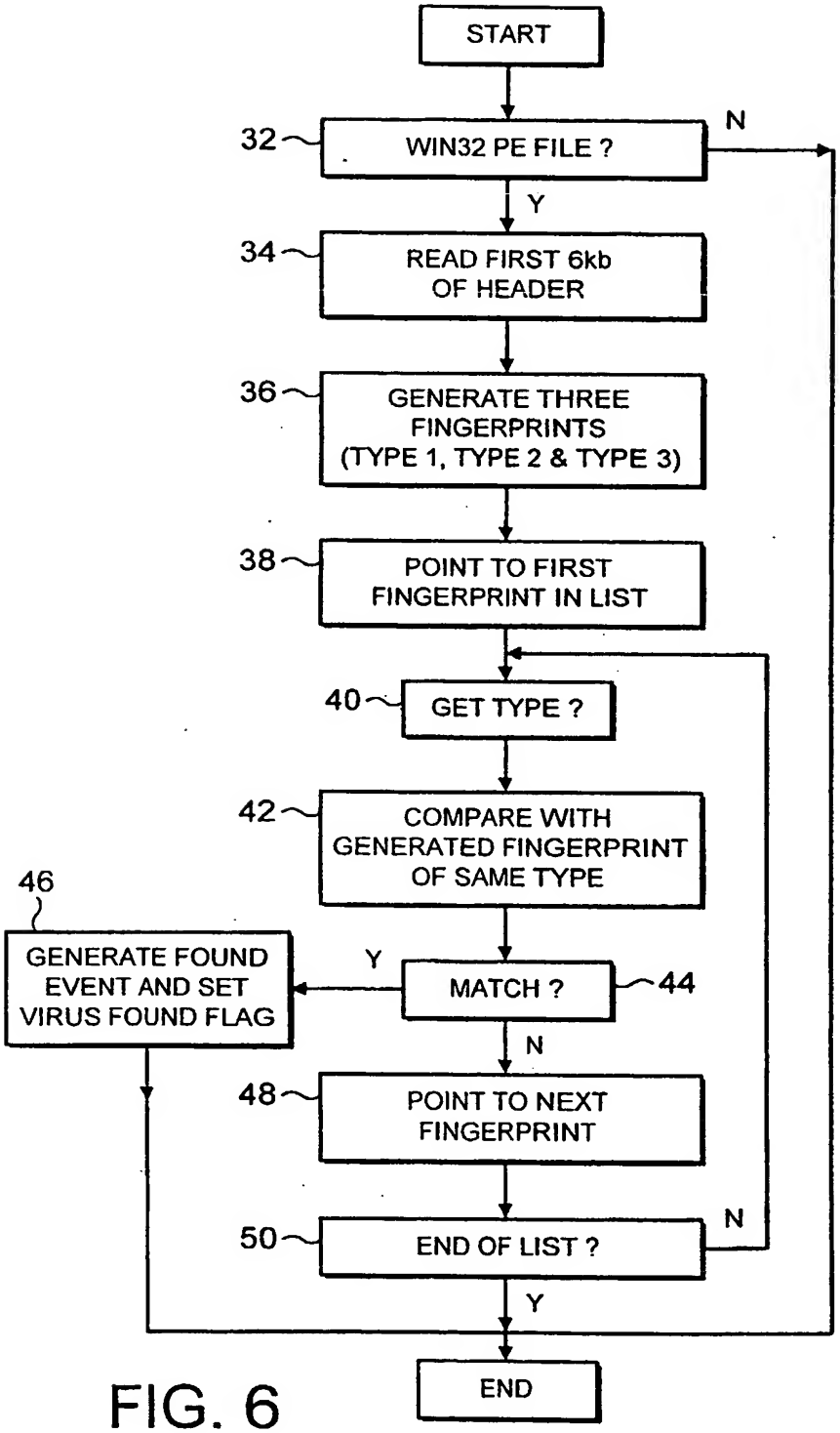

START
32
WIN32 PE FILE ?
N
Y
34
READ FIRST 6kb OF HEADER
36
GENERATE THREE FINGERPRINTS (TYPE 1, TYPE 2 & TYPE 3)
38
POINT TO FIRST FINGERPRINT IN LIST
40
GET TYPE ?
42
COMPARE WITH GENERATED FINGERPRINT OF SAME TYPE
46
GENERATE FOUND EVENT AND SET VIRUS FOUND FLAG
Y
MATCH ?
44
N
48
POINT TO NEXT FINGERPRINT
50
END OF LIST ?
N
Y
END

FIG. 6

DETECTING COMPUTER PROGRAMS WITHIN PACKED COMPUTER FILES

This invention relates to the field of data processing systems. More particularly, this invention relates to the detection of known computer programs within packed computer files.

It is known to provide Win32 packers, such as UPX, PECrypt, PECompac, Neolite and Petite, that allow a file creator to compress/encrypt an executable files' contents and add a small program stub that decompresses/decrypts the file into memory when it is run. This saves upon data storage space and the transfer of data.

A side effect of the use of such packed files is that the detection of computer programs such as Trojans and worms is made more difficult as the compression/encryption has the effect of altering the internal structure and contents of the file thereby circumventing the normal anti-virus signature scanner techniques. One approach to dealing with this problem is to incorporate functionality within the anti-virus scanner that serves to decompress and decrypt packed files before scanning them. This approach whilst effective has the drawback that it adds disadvantageous additional complexity to the scanner and increases processing load. Furthermore, each time a new packer algorithm is developed, or slightly modified, then the anti-virus scanner needs a corresponding modification that is necessarily some time period behind.

Viewed from one aspect the present invention provides a computer program product comprising a computer program operable to control a computer to detect a known computer program within a packed computer file, said packed computer file being unpacked upon execution, said computer program comprising:

resource data reading logic operable to read resource data within said packed computer file, said resource data specifying program resource items used by said known computer program and being readable by a computer operating system without dependence upon which unpacking algorithm is used by said packed computer file; and

resource data comparing logic operable to compare said resource data with characteristics of resource data of said known computer program to detect a match with said known computer program indicative of said packed computer file containing said known computer program.

The invention recognises that a packed file does not compress or encrypt the resource specifying data (e.g. a minimum subset of the resource header) as this typically needs to be accessed by the operating system of the computer when the file is being manipulated without it necessarily being executed and according unpacked. Furthermore, the independence of this resource specifying data from the pack/unpacking algorithm used allows it to be readily accessed by an anti-virus or other type of scanner. The invention also recognises that the resource specifying data is highly characteristic of the computer program to which it relates and can be used as an effective tool to identify specific known computer programs within packed files, even though the computer programs themselves may be disguised by the packing.

This ability to recognise known computer programs independent of the way in which they are packed is advantageous in a variety of circumstances, but is particularly useful when the files it is desired to detect are Trojan computer programs or worms computer programs.

Whilst the system could be provided to detect a single known computer program, it is very well suited to the type of scanner which detects any of a plurality of known computer programs within a packed computer file. This is the type of processing needed in a scanner looking for an instance of the large number of known Trojans or worms.

Whilst it would be possible to make direct comparisons between the resource data of a packed computer and the resource data for known computer programs, the efficiency, and resilience to minor changes in the resource data, is improved when the system processes the resource data to generate fingerprint data indicative of predetermined characteristics of the resource data and then comparisons are made

between the fingerprint data from a suspect packed computer file and a library of fingerprint data of known computer programs.

The resource items specified within the resource data can take a variety of forms, but typically include one or more of icon data, string data, dialog data, bitmap data, menu data and language data.

Each resource is usually specified in terms of its relative position within the computer file and the size of the resource.

The fingerprint data could be generated in many different ways and include a variety of different characteristics. In preferred embodiments the fingerprint data includes a checksum value calculated in dependence upon a number of program resource items specified between each node within a hierarchical arrangement of resource data, string names and resource sizes.

The checksum may be calculated with a rotation between the adding in of each value in order to yield some order dependence in the checksum value.

The fingerprint data may also advantageously include in preferred embodiments an indication of the number of program resource items specified within the resource data, a location of the resource item having the largest size and the size of that resource item.

As an additional or alternative feature, the fingerprint data may also include a time stamp specifying the time of compilation of the computer file within the packed computer file.

It may be that one particular selection of characteristics to be included within the fingerprint data could be insufficiently specific to the computer program concerned and accordingly the system includes the possibility of more than one type of fingerprint data

being used, the type concerned in a particular case being specified by a flag within the fingerprint data.

Whilst the invention is applicable to a wide variety of different system environments, operating systems and the like, it is particularly well suited for use when the packed computer file is a Win32 PE (portable executable) file of the type that may be executed in a Windows 95, Windows 98, Windows Millennium, Windows NT, Windows 2000 or Windows XP environment.

Complementary aspects of the invention also provide a computer program for generating characteristic data for identifying the resource data of a packed computer file together with methods and computer apparatuses in accordance with the above described techniques.

Embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 schematically illustrates a computer file being packed and unpacked;

Figure 2 schematically illustrates hierarchically arranged resource data specifying program resource items;

Figure 3 schematically illustrates the generation of a checksum value for use within fingerprint data;

Figures 4a, 4b and 4c schematically illustrate three types of fingerprint data;

Figure 5 is a flow diagram illustrating the generation of fingerprint data;

Figure 6 is a flow diagram illustrating the scanning of a packed computer file to determine if it contains one of a plurality of known computer programs; and

Figure 7 schematically illustrates a general purpose computer of the type that may be used to implement the above techniques.

Figure 1 illustrates a Win32 PE computer file of the type which may be executed by the Windows 95, Windows 98, Windows Millennium, Windows NT, Windows 2000 or Windows XP base computer systems. This computer program file 2 includes a header 4 that specifies the program resource items 6 associated with the computer program. The computer program 2 also includes executable code 8 and data 10. When the computer program 2 is packed using one of several known packing algorithms, then a packed computer file 5 is generated. In order that the operating system can properly deal with the packed computer file 5, the header 4 is not packed such that it is accessible to the operating system. The code 8, data 10 and resources 6 portions are compressed or encrypted in accordance with a packing algorithm. An unpacking computer program 12 is included within the packed computer file 5 and is executed when the packed computer file is executed so as to unpack the code 8, data 10 and resources 6 portion when execution is desired and regenerate the computer file 14 in the computer memory.

It will be appreciated that the code 8, data 10 and resources 6 may represent a computer program of any type. The computer program may be a Trojan or worm developed by a malicious person or organisation and which a user wishes to detect on their system even if it is disguised by packing. The computer program could have other forms, which it is also desired to detect, such as a computer program that is legitimate in some circumstances but that it is desired to detect in this particular situation, e.g. banned games on a business computer system.

Figure 2 schematically illustrates the hierarchical organisation of resource data within a Win32 PE file. The arrangement and format of this data is known and published in order to allow application developers to develop computer programs for use on Microsoft Corporation Windows operating systems of the above mentioned types. At the first level, different types of resources such as icons, strings, dialogues, bitmaps, menus

and the like, are defined. At a second level, one or more different instances of that particular type of program resource is specified. At the third level, one or more variants of each instance are specified with an offset value being given pointing to the location of that resource within the resource data 6 together with the size of that resource. If each of the nodes within the hierarchy is given a number specifying its order of appearance beneath the node or above it, then this three-deep hierarchy can have any point within it defined by three co-ordinates. In the example illustrated, the program resource item marked with an "*" has co-ordinates within the hierarchy of "0,0,1" and a size of "b".

Fingerprint data characteristic of the resource data of Figure 2 is generated for the purpose of rapid and robust identification of computer programs from their associated resource data. The fingerprint data can include variables such as the number of program resource items specified by the resource data and a time stamp value corresponding to the compilation time of the computer program concerned that is given within the resource specifying data at the start of that data in accordance with the known format. Another characteristic of the resource data that tends to be highly individual to a computer program is the co-ordinate position of the resource item having the largest specified size within the hierarchy together with the size value of that resource data.

A characteristic checksum value can also be calculated by parsing through the hierarchy. In particular, the checksum value can add in the number of resource items specified beneath each node within the hierarchy as the hierarchy is progressively traversed by tracking down each path to the lowest point within the hierarchy and then tracing back to the closest un-taken path and then tracking down that path. This parsing path is schematically illustrated by the dotted line in figure 2.

As well as adding in the number of items to the checksum as described above, the checksum may also add in the ASC II values of any strings naming particular resource items that are encountered during this parsing. Furthermore, the size values encountered for each resource as specified at the bottom level within the hierarchy may be added into the checksum.

Figure 3 schematically illustrates the format of the checksum value. The checksum is generated whilst "walking" the resource tree structure, and is composed of the following elements in the resource section header ...

1) The total number of entries contained in each node of the tree
2) The size of each individual resource item
3) The ASCII string name of any resource item that has a name ID.

These items are combined into the 64 bit checksum accumulator with a 32 bit XOR operation on the lower 32 bits of the checksum accumulator. After every XOR operation the checksum accumulator is rotated 1 bit to the left in order to provide an order dependence to the checksum value.

Figures 4a, 4b and 4c illustrate three different types of fingerprint values that may be employed. All three types of fingerprint are 16 bytes in length. The first byte is a control byte that indicates which type of elimination data is specified within that fingerprint. The next six bytes specify the elimination data. The following one byte specifies the number of entries processed in calculating the checksum. The final eight bytes specify the checksum itself.

Figure 4a illustrates the fingerprint format.

Figure 4b illustrates the control byte. Bits 4, 5 and 6 comprise a 3-bit field specifying the elimination data type as being one of co-ordinate/size data, timestamp data or file length data. Bits 0, 1, 2, 3, and 7 are reserved.

Figure 4c illustrates the different elimination data formats for the three different types of fingerprint. Type 1 is the primary type of fingerprint employed. Type 1 fingerprints specify the number of entries within the hierarchical resource data, the co-ordinates of the largest resource, the size of the largest resource and the checksum value.

Figure 5 illustrates the generation of fingerprint data for a known Trojan computer program. At step 16, the header 4 of the Trojan computer program is read. At step 18, this header 4 specifying the program resource items is parsed to generate the checksum value to be used within the fingerprint data and other variables such as the co-ordinates and size of the largest resource, the timestamp value of compilation and the image size are recorded. At step 20, a Type 1 fingerprint is created. At step 22, this Type 1 fingerprint is compared with a collection of fingerprints known to give false alarms as they correspond to genuine computer programs that are not Trojans and which it is not desired to detect. If step 24 indicates that the Type 1 fingerprint does not match any of the known false alarms, then the process terminates. Alternatively, processing proceeds to step 17 at which a test is made as to whether or not the fingerprint that was compared at steps 22 and 24 was a Type 1 fingerprint. If the fingerprint compared was a Type 1 fingerprint, then processing proceeds to step 18 at which a Type 2 fingerprint is generated and processing returned to step 22. If the test at step 17 indicated that a Type 1 fingerprint had not just been tested and failed, then processing proceeds to step 19. Step 19 tests as to whether or not the fingerprint just tested at steps 22 and 24 was a Type 2 fingerprint. If the fingerprint tested at steps 22 and 24 was a Type 2 fingerprint, then processing proceeds to step 21 at which a Type 3 fingerprint is generated prior to returning to step 22. If the test at step 19 indicated that the fingerprint tested at steps 22 and 24 was not a Type 2 fingerprint, then it must have been a Type 3 fingerprint and accordingly none of the fingerprints has been able to distinguish the Trojan from the known false alarm fingerprints and an error is generated at step 28.

Figure 6 is a flow diagram illustrating the scanning of a suspect packed computer file. At step 32 a determination is made as to whether the computer file is a Win32 PE file. If the computer file is not of this type, then processing terminates. If the computer file is a Win32 PE file, then step 32 serves to read the first 6kb of the suspect file, which should include all of the header resource section. This first portion of the file is stored in memory so as to be available for rapid use and processing.

The first three items in this fingerprint are indicative of the computer file concerned, but are not generally as highly specific as the checksum value. The main reason for the inclusion of the first three items within the fingerprint is to provide a mechanism for rapid elimination of fingerprints as non-matching when conducting a search through a large number of potential fingerprints. These three items of data may be used to hash into a large table of fingerprints to reduce the number of candidate fingerprints that need to be searched and thereby increase the processing speed. The checksum value tends to be highly specific to a particular collection of resource data and may be a 64-bit checksum number as discussed above or a 32-bit checksum number in less sophisticated systems.

In some circumstances the Type 1 fingerprint may not be sufficiently specific to a particular computer program. It may be that the particular computer program concerned is very simple and has few resources to characterise it, or it may be that it has been deliberately written to try and mask its uniqueness. In these circumstances, and in order to provide resistance against false alarm detections, a Type 2 fingerprint is provided for alternative use. A Type 2 fingerprint specifies a timestamp and the following two bytes in the resource data, instead of the co-ordinates of the largest resource and the size of the largest resource.

In some circumstances a Type 2 fingerprint may also not be sufficiently specific to characterise a file for elimination purposes and accordingly Type 3 fingerprints are also provided. Type 3 fingerprints specify as their elimination data four bytes corresponding to the least significant double word of the file's length. The remaining two bytes of the elimination data are 0.

The three different types of elimination data respectively specified within Type 1, Type 2 and Type 3 fingerprints provide a hierarchy of alternatives for use in providing effective elimination data. The control byte at the start of the fingerprint specifies the elimination data type and accordingly the fingerprint type.

At step 36 fingerprints of all three different types are generated from the read data and stored within memory so as to be available for rapid use. At step 38 the first fingerprint within the list of fingerprints of known computer programs it is wished to detect is selected. At step 40 the type of the fingerprint being pointed to is read. At step 42 the fingerprint being pointed to is compared with the corresponding fingerprint of the same type generated for the suspect file at step 36. Step 40 determines whether or not a match was detected at step 42. If there is a match, then step 46 generates a "found event" and the virus found flag is set such that anti-virus (in this case anti-Trojan or worm) action is initiated.

If step 44 did not detect a match, then processing proceeds to step 48 at which the next fingerprint in the list of fingerprints to be detected is pointed to. Step 50 then determines whether the end of the list of fingerprints has been reached. If the end of the list of fingerprints has not been reached, then processing is returned to step 40. If the end of the list of fingerprints has been reached, then the processing terminates.

Figure 7 illustrates a general purpose computer 200 of the type that may be used to perform the above described techniques. The general purpose computer 200 includes a central processing unit 202, a read only memory 204, a random access memory 206, a hard disk drive 208, a display driver 210 with attached display 211, a user input/output circuit 212 with attached keyboard 213 and mouse 215, a network card 214 connected to a network connection and a PC computer on a card 218 all connected to a common system bus 216. In operation, the central processing unit 202 executes a computer program that may be stored within the read only memory 204, the random access memory 206, the hard disk drive 208 or downloaded over the network card 214. Results of this processing may be displayed on the display 211 via the display driver 210. User inputs for triggering and controlling the processing are received via the user input/output circuit 212 from the keyboard 213 and mouse 215. The central processing unit 202 may use the random access 206 as its working memory. A computer program may be loaded into the computer 200 via a recording medium such as a floppy disk drive or compact disk. Alternatively, the computer program may be loaded in via the network card 214 from a remote storage drive. The PC on a card

218 may comprise its own essentially independent computer with its own working memory, CPU and other control circuitry that can co-operate with the other elements in Figure 4 via the system bus 216. The system bus 216 is a comparatively high bandwidth connection allowing rapid and efficient communication.

CLAIMS

1. A computer program product comprising a computer program operable to control a computer to detect a known computer program within a packed computer file, said packed computer file being unpacked upon execution, said computer program comprising:

resource data reading logic operable to read resource data within said packed computer file, said resource data specifying program resource items used by said known computer program and being readable by a computer operating system without dependence upon which unpacking algorithm is used by said packed computer file; and

resource data comparing logic operable to compare said resource data with characteristics of resource data of said known computer program to detect a match with said known computer program indicative of said packed computer file containing said known computer program.

2. A computer program product as claimed in claim 1, wherein said known computer program is one of:

a Trojan computer program; and

a worm computer program.

3. A computer program product as claimed any one of claims 1 and 2, wherein said resource data comparing logic is operable to compare said resource data with characteristics of a plurality of known computer programs to detect if said packed computer program contains one of said plurality of known computer programs.

4. A computer program product as claimed in any one of claims 1 to 3, wherein said resource data comparing logic is operable to processes said resource data of said packed computer file to generate fingerprint data and to compare said fingerprint data with fingerprint data of said known computer program.

5. A computer program product as claimed in any one of claims 1 to 4, wherein said program resource items used by said known computer program include one or more of:

icon data;

string data;

dialog data;

bitmap data;

menu data; and

language data.

6. A computer program product as claimed in any one of claims 1 to 5, wherein said resource data specifies for each resource item a storage location of said resource item.

7. A computer program product as claimed in claim 6, wherein said storage location of said resource item is specified as an relative offset value.

8. A computer program product as claimed in any one of claims 1 to 7, wherein said resource data specifies for each resource item a size of said resource item.

9. A computer program product as claimed in claim 4, wherein said fingerprint data includes a checksum value calculated in dependence upon one or more of:

a number of program resource items specified beneath each node within hierarchically arranged resource data;

string names associated with program resource items within said resource data; and

sizes of program resource items within said resource data.

10. A computer program product as claimed in claim 4, wherein said fingerprint data includes a number of program resource items specified within said resource data.

11. A computer program product as claimed in claim 4, wherein said fingerprint data includes a location within said resource data of an entry specifying a program resource item having a largest size.

12. A computer program product as claimed in claim 4, wherein said fingerprint data includes timestamp data indicative of a time of compilation of said known computer program.

13. A computer program product as claimed in claim 4, wherein said fingerprint data includes a flag indicating which data is included within said fingerprint data.

14. A computer program product as claimed in claim 9, wherein said checksum value is rotated between each item being added into said checksum.

15. A computer program product as claimed in any one of claims 1 to 14, wherein said packed computer file includes an unpacking computer program which upon execution decompresses said known computer program.

16. A computer program product as claimed in any one of claims 1 to 15, wherein said packed computer file is a Win32 PE file.

17. A computer program product comprising a computer program operable to control a computer to generate data for detecting a known computer program within a packed computer file, said packed computer file being unpacked upon execution, said computer program comprising:

resource data reading logic operable to read resource data within said packed computer file, said resource data specifying program resource items used by said known computer program and being readable by a computer operating system without dependence upon which unpacking algorithm is used by said packed computer file; and

characteristic data generating logic operable to generate characteristic data associated with said resource data for comparison with characteristics of resource data of

said known computer program to detect a match with said known computer program indicative of said packed computer file containing said known computer program.

18. A computer program product as claimed in claim 17, wherein said known computer program is one of:

a Trojan computer program; and

a worm computer program.

19. A computer program product as claimed in any one of claims 17 and 18, wherein said characteristic data generating logic is operable to generate characteristic data from a plurality of known computer programs to enable detection of any of said plurality of known computer programs within said packed computer program..

20. A computer program product as claimed in any one of claims 17 to 19, wherein said characteristic data generating logic is operable to processes said resource data of said packed computer file to generate fingerprint data for comparison with fingerprint data generated from packed computer files.

21. A computer program product as claimed in any one of claims 17 to 20, wherein said program resource items used by said known computer program include one or more of:

icon data;

string data;

dialog data;

bitmap data;

menu data; and

language data.

22. A computer program product as claimed in any one of claims 17 to 21, wherein said resource data specifies for each resource item a storage location of said resource item.

23. A computer program product as claimed in claim 22, wherein said storage location of said resource item is specified as an relative offset value.

24. A computer program product as claimed in any one of claims 17 to 23, wherein said resource data specifies for each resource item a size of said resource item.

25. A computer program product as claimed in claim 20, wherein said fingerprint data includes a checksum value calculated in dependence upon one or more of:

a number of program resource items specified beneath each node within hierarchically arranged resource data;

string names associated with program resource items within said resource data; and

sizes of program resource items within said resource data.

26. A computer program product as claimed in claim 20, wherein said fingerprint data includes a number of program resource items specified within said resource data.

27. A computer program product as claimed in claim 20, wherein said fingerprint data includes a location within said resource data of an entry specifying a program resource item having a largest size.

28. A computer program product as claimed in claim 20, wherein said fingerprint data includes timestamp data indicative of a time of compilation of said known computer program.

29. A computer program product as claimed in claim 20, wherein said fingerprint data includes a flag indicating which data is included within said fingerprint data.

30. A computer program product as claimed in claim 25, wherein said checksum value is rotated between each item being added into said checksum.

31. A computer program product as claimed in any one of claims 17 to 30, wherein said packed computer file includes an unpacking computer program which upon execution decompresses said known computer program.

32. A computer program product as claimed in any one of claims 17 to 31, wherein said packed computer file is a Win32 PE file.

33. A method of controlling a computer to detect a known computer program within a packed computer file, said packed computer file being unpacked upon execution, said method comprising the steps of:

reading resource data within said packed computer file, said resource data specifying program resource items used by said known computer program and being readable by a computer operating system without dependence upon which unpacking algorithm is used by said packed computer file; and

comparing said resource data with characteristics of resource data of said known computer program to detect a match with said known computer program indicative of said packed computer file containing said known computer program.

34. A method as claimed in claim 33, wherein said known computer program is one of:

a Trojan computer program; and

a worm computer program.

35. A method as claimed in any one of claims 33 and 34, wherein said step of comparing compares said resource data with characteristics of a plurality of known computer programs to detect if said packed computer program contains one of said plurality of known computer programs.

36. A method as claimed in any one of claims 33 to 35, comprising processing said resource data of said packed computer file to generate fingerprint data and comparing said fingerprint data with fingerprint data of said known computer program.

37. A method as claimed in any one of claims 33 to 36, wherein said program resource items used by said known computer program include one or more of:

icon data;

string data;

dialog data;

bitmap data;

menu data; and

language data.

38. A method as claimed in any one of claims 33 to 37, wherein said resource data specifies for each resource item a storage location of said resource item.

39. A method as claimed in claim 38, wherein said storage location of said resource item is specified as an relative offset value.

40. A method as claimed in any one of claims 33 to 39, wherein said resource data specifies for each resource item a size of said resource item.

41. A method as claimed in claim 36, wherein said fingerprint data includes a checksum value calculated in dependence upon one or more of:

a number of program resource items specified beneath each node within hierarchically arranged resource data;

string names associated with program resource items within said resource data; and

sizes of program resource items within said resource data.

42. A method as claimed in claim 36, wherein said fingerprint data includes a number of program resource items specified within said resource data.

43. A method as claimed in claim 36, wherein said fingerprint data includes a location within said resource data of an entry specifying a program resource item having a largest size.

44. A method as claimed in claim 36, wherein said fingerprint data includes timestamp data indicative of a time of compilation of said known computer program.

45. A method as claimed in claim 36, wherein said fingerprint data includes a flag indicating which data is included within said fingerprint data.

46. A method as claimed in claim 41, wherein said checksum value is rotated between each item being added into said checksum.

47. A method as claimed in any one of claims 33 to 46, wherein said packed computer file includes an unpacking computer program which upon execution decompresses said known computer program.

48. A method as claimed in any one of claims 33 to 47, wherein said packed computer file is a Win32 PE file.

49. A method of controlling a computer to generate data for detecting a known computer program within a packed computer file, said packed computer file being unpacked upon execution, said method comprising the steps of:

reading resource data within said packed computer file, said resource data specifying program resource items used by said known computer program and being readable by a computer operating system without dependence upon which unpacking algorithm is used by said packed computer file; and

generating characteristic data associated with said resource data for comparison with characteristics of resource data of said known computer program to detect a match with said known computer program indicative of said packed computer file containing said known computer program.

50. A method as claimed in claim 49, wherein said known computer program is one of:

a Trojan computer program; and

a worm computer program.

51. A method as claimed in any one of claims 49 and 50, wherein said step of generating generates characteristic data from a plurality of known computer programs to enable detection of any of said plurality of known computer programs within said packed computer program..

52. A method as claimed in any one of claims 49 to 51, comprising processing said resource data of said packed computer file to generate fingerprint data for comparison with fingerprint data generated from packed computer files.

53. A method as claimed in any one of claims 49 to 52, wherein said program resource items used by said known computer program include one or more of:

icon data;

string data;

dialog data;

bitmap data;

menu data; and

language data.

54. A method as claimed in any one of claims 49 to 53, wherein said resource data specifies for each resource item a storage location of said resource item.

55. A method as claimed in claim 54, wherein said storage location of said resource item is specified as an relative offset value.

56. A method as claimed in any one of claims 49 to 55, wherein said resource data specifies for each resource item a size of said resource item.

57. A method as claimed in claim 52, wherein said fingerprint data includes a checksum value calculated in dependence upon one or more of:

a number of program resource items specified beneath each node within hierarchically arranged resource data;

string names associated with program resource items within said resource data; and

sizes of program resource items within said resource data.

58. A method as claimed in claim 52, wherein said fingerprint data includes a number of program resource items specified within said resource data.

59. A method as claimed in claim 52, wherein said fingerprint data includes a location within said resource data of an entry specifying a program resource item having a largest size.

60. A method as claimed in claim 52, wherein said fingerprint data includes timestamp data indicative of a time of compilation of said known computer program.

61. A method as claimed in claim 52, wherein said fingerprint data includes a flag indicating which data is included within said fingerprint data.

62. A method as claimed in claim 57, wherein said checksum value is rotated between each item being added into said checksum.

63. A method as claimed in any one of claims 49 to 62, wherein said packed computer file includes an unpacking computer program which upon execution decompresses said known computer program.

64. A method as claimed in any one of claims 49 to 63, wherein said packed computer file is a Win32 PE file.

65. Apparatus for detecting a known computer program within a packed computer file, said packed computer file being unpacked upon execution, said apparatus comprising:

a resource data reader operable to read resource data within said packed computer file, said resource data specifying program resource items used by said known computer program and being readable by a computer operating system without dependence upon which unpacking algorithm is used by said packed computer file; and

a resource data comparitor operable to compare said resource data with characteristics of resource data of said known computer program to detect a match with said known computer program indicative of said packed computer file containing said known computer program.

66. Apparatus as claimed in claim 65, wherein said known computer program is one of:

a Trojan computer program; and

a worm computer program.

67. Apparatus as claimed in any one of claims 65 and 66, wherein said resource data comparitor is operable to compare said resource data with characteristics of a plurality of known computer programs to detect if said packed computer program contains one of said plurality of known computer programs.

68. Apparatus as claimed in any one of claims 65 to 67, wherein said resource data comparitor is operable to processes said resource data of said packed computer file to

generate fingerprint data and to compare said fingerprint data with fingerprint data of said known computer program.

69. Apparatus as claimed in any one of claims 65 to 68, wherein said program resource items used by said known computer program include one or more of:

icon data;

string data;

dialog data;

bitmap data;

menu data; and

language data.

70. Apparatus as claimed in any one of claims 65 to 69, wherein said resource data specifies for each resource item a storage location of said resource item.

71. Apparatus as claimed in claim 70, wherein said storage location of said resource item is specified as an relative offset value.

72. Apparatus as claimed in any one of claims 65 to 71, wherein said resource data specifies for each resource item a size of said resource item.

73. Apparatus as claimed in claim 68, wherein said fingerprint data includes a checksum value calculated in dependence upon one or more of:

a number of program resource items specified beneath each node within hierarchically arranged resource data;

string names associated with program resource items within said resource data; and

sizes of program resource items within said resource data.

74. Apparatus as claimed in claim 68, wherein said fingerprint data includes a number of program resource items specified within said resource data.

75. Apparatus as claimed in claim 68, wherein said fingerprint data includes a location within said resource data of an entry specifying a program resource item having a largest size.

76. Apparatus as claimed in claim 68, wherein said fingerprint data includes timestamp data indicative of a time of compilation of said known computer program.

77. Apparatus as claimed in claim 68, wherein said fingerprint data includes a flag indicating which data is included within said fingerprint data.

78. Apparatus as claimed in claim 73, wherein said checksum value is rotated between each item being added into said checksum.

79. Apparatus as claimed in any one of claims 65 to 78, wherein said packed computer file includes an unpacking computer program which upon execution decompresses said known computer program.

80. Apparatus as claimed in any one of claims 65 to 79, wherein said packed computer file is a Win32 PE file.

81. Apparatus for generating data for detecting a known computer program within a packed computer file, said packed computer file being unpacked upon execution, said apparatus comprising:

a resource data reader operable to read resource data within said packed computer file, said resource data specifying program resource items used by said known computer program and being readable by a computer operating system without dependence upon which unpacking algorithm is used by said packed computer file; and

a characteristic data generator operable to generate characteristic data associated with said resource data for comparison with characteristics of resource data of said

known computer program to detect a match with said known computer program indicative of said packed computer file containing said known computer program.

82. Apparatus as claimed in claim 81, wherein said known computer program is one of:

a Trojan computer program; and

a worm computer program.

83. Apparatus as claimed in any one of claims 81 and 82, wherein said characteristic data generator is operable to generate characteristic data from a plurality of known computer programs to enable detection of any of said plurality of known computer programs within said packed computer program..

84. Apparatus as claimed in any one of claims 81 to 83, wherein said characteristic data generator is operable to processes said resource data of said packed computer file to generate fingerprint data for comparison with fingerprint data generated from packed computer files.

85. Apparatus as claimed in any one of claims 81 to 84, wherein said program resource items used by said known computer program include one or more of:

icon data;

string data;

dialog data;

bitmap data;

menu data; and

language data.

86. Apparatus as claimed in any one of claims 81 to 85, wherein said resource data specifies for each resource item a storage location of said resource item.

87. Apparatus as claimed in claim 86, wherein said storage location of said resource item is specified as an relative offset value.

88. Apparatus as claimed in any one of claims 81 to 87, wherein said resource data specifies for each resource item a size of said resource item.

89. Apparatus as claimed in claim 84, wherein said fingerprint data includes a checksum value calculated in dependence upon one or more of:

a number of program resource items specified beneath each node within hierarchically arranged resource data;

string names associated with program resource items within said resource data; and

sizes of program resource items within said resource data.

90. Apparatus as claimed in claim 84, wherein said fingerprint data includes a number of program resource items specified within said resource data.

91. Apparatus as claimed in claim 84, wherein said fingerprint data includes a location within said resource data of an entry specifying a program resource item having a largest size.

92. Apparatus as claimed in claim 84, wherein said fingerprint data includes timestamp data indicative of a time of compilation of said known computer program.

93. Apparatus as claimed in claim 84, wherein said fingerprint data includes a flag indicating which data is included within said fingerprint data.

94. Apparatus as claimed in claim 89, wherein said checksum value is rotated between each item being added into said checksum.

95. Apparatus as claimed in any one of claims 81 to 94, wherein said packed computer file includes an unpacking computer program which upon execution decompresses said known computer program.

96. Apparatus as claimed in any one of claims 81 to 95, wherein said packed computer file is a Win32 PE file.

97. A computer program product comprising a computer program operable to control a computer to detect known computer program within packed computer file, said packed computer file being unpacked upon execution substantially as hereinbefore described with reference to the accompanying drawings.

98. A computer program product comprising a computer program operable to control a computer to generate data for detecting a known computer program within a packed computer file, said packed computer file being unpacked upon execution substantially as hereinbefore described with reference to the accompanying drawings.

99. A method of controlling a computer to detect a known computer program within a packed computer file, said packed computer file being unpacked upon execution substantially as hereinbefore described with reference to the accompanying drawings.

100. A method of controlling a computer to generate data for detecting a known computer program within a packed computer file, said packed computer file being unpacked upon execution substantially as hereinbefore described with reference to the accompanying drawings.

101. Apparatus for detecting a known computer program within a packed computer file, said packed computer file being unpacked upon execution substantially as hereinbefore described with reference to the accompanying drawings.

102. Apparatus for generating data for detecting a known computer program within a packed computer file, said packed computer file being unpacked upon execution substantially as hereinbefore described with reference to the accompanying drawings.

Application No: GB 0129835.5 291
Claims searched: 1 to 102

Examiner: Nik Dowell
Date of search: 19 June 2002

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.T): G4A (AAP, AFMX)

Int Cl (Ed.7): G06F (1/00)

Other: Online : WPI,EPODOC,PAJ,INSPEC,ELSEVIER,TDB

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	GB 2 365 158 A	(Content Technologies) See whole document	1, 17, 33, 49, 65 and 81 at least
X	US 5 991 714	(Shanner) See especially column 5, lines 52 to 61	1, 17, 33, 49, 65 and 81 at least

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.